TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: M2612 – Elektrotechnika a informatika
Studijní obor: 3902T005 – Automatické řízení a inženýrská informatika

Vytvoření knihovny pro podporu rozpoznávání objektu v obraze

Creation of library supporting image object recognition

Diplomová práce

Autor: **Jan Zíma**
Vedoucí práce: Ing. Jaroslav Buchta

V Liberci 17. 12. 2007

/* tady bude zadání*/

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Abstrakt

Diplomová práce se zabývá předzpracováním a segmentací obrazu. Popisuje základní i pokročilé algoritmy, které jsou dílčí součástí celého procesu zpracování obrazové informace. Veškeré uvedené metody jsou nejprve vysvětleny z hlediska teoretického, a následně jsou naznačeny možnosti jejich realizace. Nechybí rovněž metodika programování dynamicky linkovaných knihoven.

Praktickým cílem této práce je naprogramovat dynamicky linkovanou knihovnu a do ní implementovat některé vybrané metody. Tyto implementované algoritmy jsou porovnány z hlediska časové náročnosti s jinými volně dostupnými prostředky. Rozhraní knihovny a exportované funkce jsou na závěr patřičně zdokumentovány.

Klíčová slova:

zpracování obrazu, segmentace, dynamicky linkovaná knihovna, hledání hran, matematická morfologie.

Abstract

This diploma thesis deals with a preprocessing and segmentation of image. It describes simple and advanced algorithms, that are parts of whole process of image processing. First of all the methods are explained in term of theoretical and then there are their implementation possibilities demonstrated. A guidance for dynamic link libraries programming is also provided.

The practical aim of this diploma thesis is a dynamic link library programming and an implementation of some chosen methods into it. These implemented algorithms are compared in term of time-consuming with other free available resources. Finally the interface of the library and the exported functions are thoroughly documented.

Keywords:

image processing, segmentation, dynamic link library, edge detection, mathematical morphology.

Obsah

Seznam použitých symbolů a zkratek

$f(x, y)$	výstupní obraz
$g(x, y)$	vstupní obraz
x, y	diskrétní souřadnice
X, Y	spojité souřadnice
M, N	rozměry obrazu
H	histogram
h	konvoluční maska
T	vyhledávací tabulka
n	rozměr okolí
i, j	indexy polí
σ	střední kvadratická odchylka
G	Gaussián
$\langle p_{MIN}, p_{MAX} \rangle$	vstupní interval jasů
$\langle q_{MIN}, q_{MAX} \rangle$	výstupní interval jasů
T_x, T_y	transformační vztahy
a_i, b_i	koeficienty polynomů
exp	exponenciální funkce
∇	gradient
∇^2	Laplaceův operátor (Laplacián)
Ψ	směr gradientu
Δ	diference
∂	parciální derivace
$*$	diskrétní konvoluce
A	vstupní obraz (morfologie)
B	strukturní element
$\oplus$	dilatace
$\otimes$	tref či miň
Θ	eroze
$\bullet$	uzavření
$\circ$	otevření

např.	například
popř.	popřípadě
resp.	respektive
obr.	obrázek
atd.	a tak dále
apod.	a podobně
tzv.	tak zvaně
tj.	to jest

Poznámka: významy uvedených symbolů platí, pokud není v textu uvedeno jinak.

Úvod

Zpracování obrazu, neboli počítačové vidění, lze rozdělit do těchto základních kroků: snímání a digitalizace obrazu; předzpracování; segmentace obrazu na objekty; popis objektů; jejich klasifikace či porozumění obsahu obrazu. Tato práce se zabývá především metodikou předzpracování a segmentace, a to jak po stránce teoretické, tak i praktické. V praktické části jde o realizaci vybraných algoritmů v podobě dynamicky linkované knihovny. Hlavním úkolem těchto implementovaných metod je tedy úprava obrazových dat před dalším zpracováním, tj. rozpoznáváním objektů.

První kapitola popisuje metody předzpracování. Jsou zde vysvětleny základní i složitější postupy jako např. filtrace, detekce hran nebo matematická morfologie.

Druhá kapitola pojednává o nejznámějších segmentačních postupech, mezi které patří prahování, sledování hranice, Houghova transformace či narůstání oblastí.

Třetí kapitola se zabývá tvorbou dynamicky linkovaných knihoven v prostředí Microsoft Visual C++.

Kapitola čtvrtá pak poskytuje dokumentaci k vytvořené knihovně, jejímu rozhraní, exportovaným funkcím a jednoduché aplikaci, ve které lze tuto knihovnu otestovat.

Zhodnocení práce prostřednictvím porovnání této knihovny s jinými, volně dostupnými nástroji, má za úkol poslední, pátá kapitola. Jednotlivé algoritmy jsou posuzovány z hlediska časové náročnosti.

1. Předzpracování obrazu

1.1 Obecně

Hlavním úkolem předzpracování je potlačení rušivých vlivů vzniklých při pořízení a digitalizaci obrazu. Jedná se hlavně o šum a zkreslení. Dalším cílem je také zvýraznění určitých vlastností, kterých můžeme využít při další práci s obrazem. Obrazová data jsou ve formě obdélníkových matic, které reprezentují diskrétní obrazovou funkci. Obrazová funkce nabývá celočíselných hodnot, odpovídajících jasovým úrovním.

Předzpracováním se snažíme obraz upravit tak, aby následné rozpoznávání objektů bylo co možná nejjednodušší. Je zřejmé, že do obrazu nevnášíme žádnou novou informaci, spíše dochází ke značné redukci velkého množství obrazových dat. Je-li množství informace obsažené v obraze nedostatečné, je nutno zkvalitnit způsob jeho pořízení.

1.2 Bodové jasové transformace

1.2.1 Jasové korekce

Hlavním úkolem jasových korekcí je odstranit poruchy způsobené nestejnou citlivostí snímacích prvků kamery v různých částech obrazu. Tato vada je v praxi způsobena zeslabeným světlem, které prochází dále od optické osy. Výsledný jas obrazového elementu je závislý pouze na jasu elementu o stejných souřadnicích v původním obraze.

Podle [1] lze jasovými korekcemi potlačit systematické poruchy v případě, že známe odchylku citlivosti každého bodu obrazu od ideální převodní charakteristiky. Porušení obrazu aproximujeme multiplikativním koeficientem $e(x, y)$. Zkreslený obraz $f(x, y)$ je dán vztahem

$$f(x, y) = e(x, y)g(x, y). \tag{1.1}$$

Pomocí obrazu $f_c(x, y)$ (etalonu o konstantním jasu *c*) určíme degradační transformaci *e*. Poruchy pak odstraníme podle vztahu

$$g(x, y) = \frac{f(x, y)}{e(x, y)} = \frac{cf(x, y)}{f_c(x, y)}. \tag{1.2}$$

Uvedenou metodu lze použít jen při stálých snímacích podmínkách. Dochází-li ke změnám podmínek, je nutno po každém kalibrování snímacího prvku stanovit novou korekční transformační matici.

Při výpočtu výstupních obrazových hodnot se může stát, že výsledné hodny se ocitnou mimo interval přípustných jasových úrovní. Tyto nepřípustné jasové úrovně je třeba nahradit krajními hodnotami intervalu jasové stupnice.

1.2.2 Modifikace jasové stupnice

Při těchto operacích dochází k transformaci celé jasové stupnice. Výstupní hodnoty obrazových bodů tedy nezávisí na jejich poloze ve vstupním obraze. Cílem těchto metod je především usnadnit člověku interpretaci obrazových dat, jako například zvýšení kontrastu u rentgenových snímků. Pro automatickou analýzu obrazu však nejsou tyto transformace příliš vhodné, protože mohou v některých případech vést ke ztrátě obrazové informace.

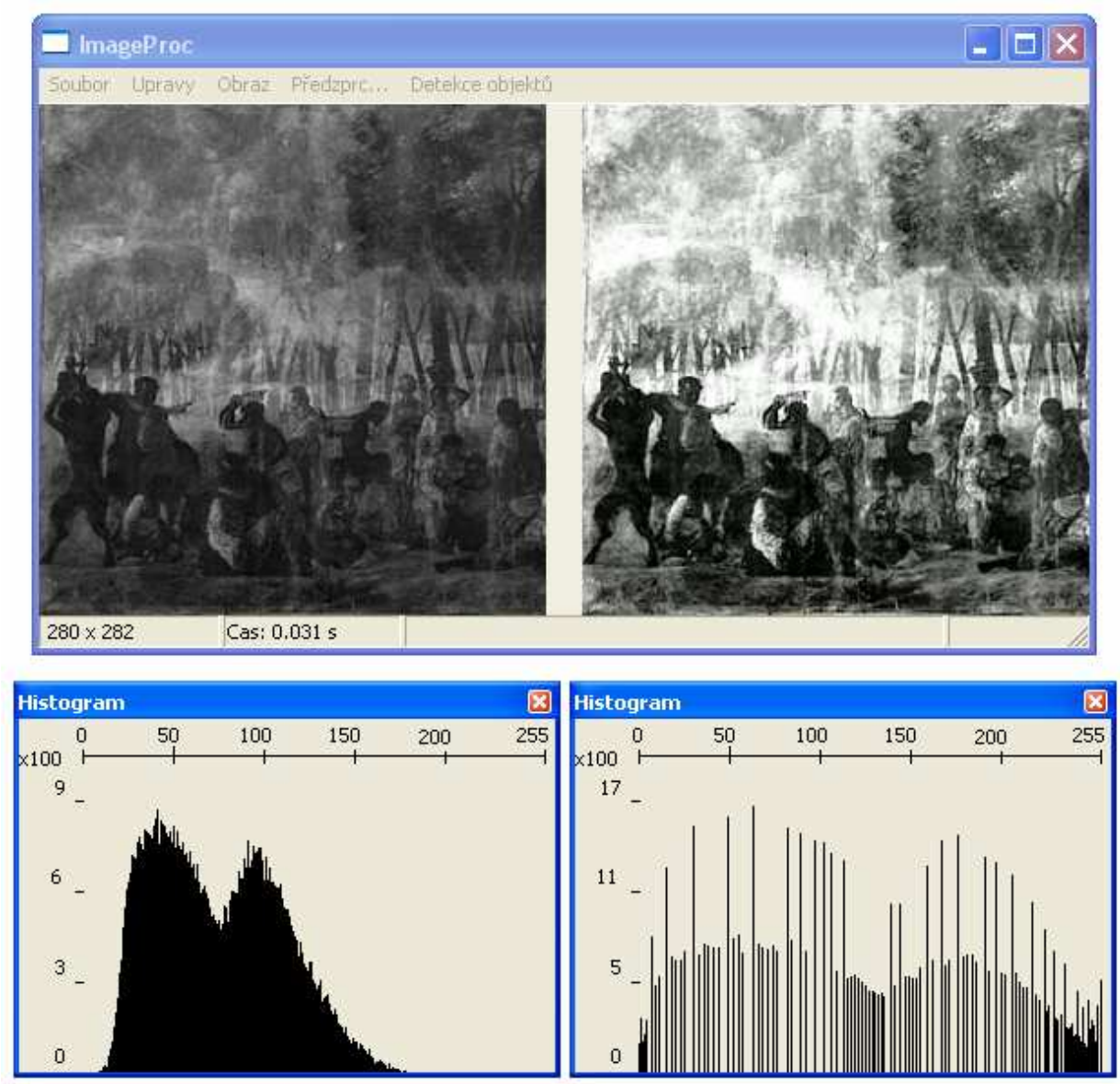
Modifikaci jasové stupnice lze snadno implementovat pomocí vyhledávací tabulky (look up table). Počet položek tabulky odpovídá počtu jasových úrovní v obraze, přičemž jednotlivé položky obsahují nové hodnoty jasu, vypočtené pomocí transformace. Původní hodnoty jasu jsou při výpočtu použity jako indexy do vyhledávací tabulky. Výsledek je tedy možné pozorovat v reálném čase. Při zpracování barevných obrazů je tabulka vytvořena pro každou barevnou složku.

V praxi je asi nejpoužívanější metodou **vyrovnání (ekvalizace) histogramu**, kdy dochází podle [1] ke zvýšení kontrastu v oblasti maximálních hodnot histogramu a snížení v okolí minimálních. Zastoupení jednotlivých jasových úrovní v obraze je pak zhruba stejné.

Hodnoty ve vyhledávací tabulce vypočteme podle vztahu:

$$T(p) = \frac{q_{MAX} - q_{MIN}}{N \cdot M} \sum_{i=p_{MIN}}^{p} H(i) + q_{MIN}, \tag{1.3}$$

kde $\langle q_{MIN}, q_{MAX} \rangle$ je výstupní interval jasů, $\langle p_{MIN}, p_{MAX} \rangle$ je vstupní interval jasů, *N* a *M* jsou rozměry obrazu a *H(i)* je kumulativní histogram, který vypočteme z normálního histogramu. Někdy se také používá logaritmická transformace, protože lépe odpovídá citlivosti lidského oka.



Obr. 1.1 – Ekvalizace histogramu

Roztažení (normalizace) histogramu lze použít tehdy, je-li interval jasů zastoupených v obraze menší než rozsah jasových hodnot (většinou 0 až 255). Prvnímu nenulovému prvku histogramu je přiřazen index 0, poslednímu pak index *i-1*. Nové hodnoty jasů pak získáme pomocí vyhledávací tabulky, kterou vypočteme podle vztahu:

$$T(p) = \frac{q_{MAX} - q_{MIN}}{p_{MAX} - p_{MIN}} (p - p_{MIN}) + q_{MAX} . \qquad (1.4)$$

1.3 Geometrické transformace

Při snímání obrazu může často docházet ke geometrickým zkreslením. Zkreslení může být způsobeno například vzdalující se kamerou od objektu, otáčením zeměkoule při fotografování z vesmíru, a nebo spoustou jiných příčin. Geometrických transformací

nevyužíváme jen pro korekce zkreslení, ale také při operacích jako je rotace, zkosení či změna velikosti. Jedná se v podstatě o vektorovou funkci (obecný vztah (1.5)), která zobrazí bod (x, y) z původního obrazu do bodu (x', y') ve výstupním obraze.

$$x' = T_x(x, y), \qquad y' = T_y(x, y). \tag{1.5}$$

Tyto transformační vztahy je možné sestrojit na základě znalosti vstupního a výstupního obrazu, a to za pomoci několika známých vlícovacích bodů, které představují tentýž objekt v obou obrazech. V případě rotace, změna měřítka a podobných transformací jsou transformační rovnice předem známé. Takto definovaná geometrická transformace se skládá ze dvou kroků, transformace souřadnic bodů a následná aproximace jasové stupnice.

Transformace souřadnic bodů má za úkol nalézt k bodu v původním obraze odpovídající bod ve výstupním obraze. Výsledné souřadnice jsou zpravidla neceločíselné a určíme je rovnic (1.6), což jsou aproximace obecných vztahů (1.5).

$$x' = \sum_{r=0}^{m}\sum_{k=0}^{m-r} a_{rk} x^r y^k, \qquad y' = \sum_{r=0}^{m}\sum_{k=0}^{m-r} b_{rk} x^r y^k \ . \tag{1.6}$$

Koeficienty a_{rd} a b_{rk} vypočteme metodou nejmenších čtverců, pomocí dvojic vlícovacích bodů. V praxi však často postačí (1.6) aproximovat bilineární transformací (1.7) nebo afinní transformací (1.8). Toto nahrazení provádíme zejména tehdy, je-li geometrická transformace podobná po celé ploše obrazu.

$$\begin{aligned} x' &= a_0 + a_1 x + a_2 y + a_3 xy \\ y' &= b_0 + b_1 x + b_2 y + b_3 xy \end{aligned} \tag{1.7}$$

$$\begin{aligned} x' &= a_0 + a_1 x + a_2 y \\ y' &= b_0 + b_1 x + b_2 y \end{aligned} \tag{1.8}$$

Složitější geometrické transformace lze realizovat, že obraz rozdělíme na menší oblasti a pro každou oblast použijeme jinou – jednodušší transformaci.

Při **aproximaci jasové stupnice** se jedná v podstatě o interpolaci jasové funkce. V předchozím kroku jsme získali neceločíselné souřadnice bodů ve výstupním obraze a teď je třeba k nim nalézt jasové úrovně. Kvalitu výsledného obrazu ovlivní přesnost interpolace, čili velikost okolí, ze kterého budeme interpolovat.

Asi nejednoduší metodou je **interpolace nejbližším sousedem**, kdy výstupnímu bodu je přiřazen jas nebližšího bodu ve vstupním obraze. Implementace je velmi rychlá a jednoduchá, výsledný jas získáme podle vztahu:

$$f(x, y) = g(round(x), round(y)). \tag{1.9}$$

Chyba při interpolaci je maximálně půl pixelu, což vede k nežádoucím efektům jako např. mizení slabých hran při zmenšování, nebo zvýraznění skoků na hranách s malým sklonem při zvětšování.

Další poměrně rychlou metodou je **bilineární interpolace**, protože pro výpočet využívá pouze čtyř bodů ze svého okolí. Jas výstupního pixelu je dán lineární kombinací bodů z okolí a je úměrný jejich vzdálenosti od zpracovávaného bodu, což je vidět ze vztahu (1.10). Jako nevýhodu lze uvést rozmazávání ostrých obrysů.

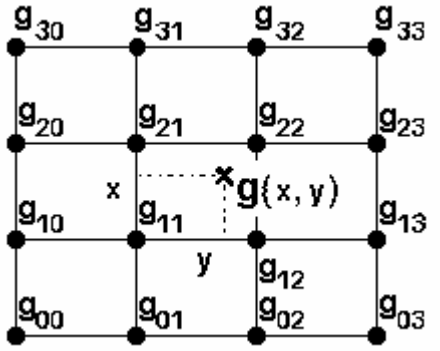
$$f(x,y) = (1-a)(1-b)g(l,k) + a(1-b)g(l+1,k)b(1-a)g(l,k+1) + ab \cdot g(l+1,k+1) \quad (1.10)$$

kde $l = round(x), \quad a = x - l,$
$k = round(y), \quad b = y - k.$

K ještě přesnějším nástrojům patří **interpolace bikubická**. Obrazová funkce je aproximována jako bikubický polynom. Jas vypočteme z okolí obsahující 16 obrazových bodů podle vztahu (1.11). Hlavní výhodou je malé rozmazávání hran, nevýhodou je pak větší časová náročnost o proti výše zmíněným metodám.

$$f(x,y) = \sum_{i,j=0}^{3} C_i(x) C_j(y) g(x,y), \quad (1.11)$$

kde C_i a C_j jsou kubické polynomy.



Obr. 1.2 – Zpracovávané okolí při bikubické interpolaci

1.4 Lokální předzpracování

Metody lokálního předzpracování využívají pro výpočet nového jasu bodu jen lokální okolí příslušného vstupního bodu v obraze. Ve většině případů nevyužívají předběžné znalosti obrazu. Tyto metody můžeme dále rozdělit do dvou skupin. **Vyhlazováním** obrazu potlačíme vyšší frekvence obrazové funkce a tím i eliminujeme náhodný šum. Nevýhodou těchto operací je to, že při nich dochází k eliminaci ostatních náhlých změn obrazové funkce, jako např. hrany a tenké linie. Druhou skupinou pak jsou **gradientní operace**, pomocí nichž je možné zdůraznit vyšší frekvence a tím obraz

zostřit. Současně dojde také ke zvýraznění hran v obraze. Negativním důsledkem je, ale i zvýraznění šumu. Někdy je tedy výhodné použít algoritmy, které v sobě kombinují oba uvedené postupy.

1.4.1 Filtrace

Úkolem filtrace je především potlačení velkých rozdílů jasů uvnitř menších oblastí, tedy šum. To odpovídá potlačení vysokých plošných frekvencí ve frekvenční oblasti (plošné frekvence jsou výsledkem dvourozměrné Fourierovy transformace obrazu). Volba metody a velikosti filtrovaného okolí bodu závisí na velikosti významných detailů v obraze a dále pak na druhu šumu, kterým je obraz zatížen. K nevýhodou patří to, že při filtraci dochází k rozmazání hran.

Asi nejjednodušším postupem je **filtrace prostým průměrem**. Vycházíme z předpokladu, že sousední obrazové body mají podobnou hodnotu jasu. Při obyčejném průměrování filtrujeme obraz tím, že obrazovému bodu přiřadíme novou hodnotu v podobě průměru jasu z jejího čtvercového okolí. Velikost okolí by však neměla být větší než je velikost nejmenšího důležitého detailu v obraze. Touto metodou lze odstranit skvrny šumu menší než velikost okolí. Nevýhodou prostého průměrování je velké rozmazávání hran, proto se tato metoda používá spíše jako pomocná při jiných složitějších metodách filtrace. Rozmazání hran můžeme částečně eliminovat tím, že povolíme pouze určitý rozsah přípustných rozdílů mezi původním jasem bodu a nově vypočtené hodnoty. V případě, že je rozdíl větší než předem zvolený práh, použije se původní hodnota jasu. V opačném případě je použita nová hodnota. Výhodou metody obyčejného průměrování je jednoduchá realizace.

Výstupní hodnoty jasů vypočteme podle vztahu:

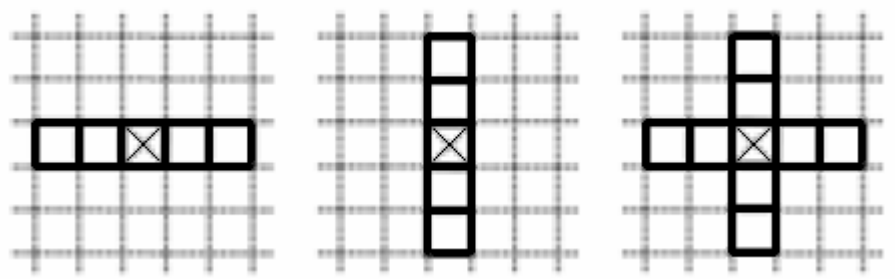
$$f(x,y)=\frac{1}{(2n+1)^2}\sum_{i=-n}^{n}\sum_{j=-n}^{n}g(x+i,y+j), \tag{1.12}$$

kde n je rozměr okolí (např. když $n = 1$ jedná se o okolí 3x3), $g(x,y)$ je vstupní obraz, $f(x,y)$ je výstupní obraz.

Máme-li k dispozici více obrazů téhož předmětu, nefiltrujeme na okolí, ale k výpočtu výstupního jasu použijeme body na stejných souřadnicích v různých obrazech. Tak je možné dosáhnout dobrých výsledků filtrace bez rozmazání hran.

Podobnou metodou je **filtrace pomocí mediánu**. Nový jas obrazového bodu je stanoven jako medián, určený z hodnot příslušného okolí bodu v původním obraze. Pro nalezení mediánu tedy musíme seřadit body, které jsou obsaženy v okolí, podle

velikosti jasu. Rozmazání hran není tak výrazné jako u prostého průměru a impulsní šum je celkem dobře potlačen. Jako další výhodu zle uvést použitelnost při vyhlazování binárních obrazů, kde by výsledkem filtrace pomocí ostatních metod, byly nepřípustné hodnoty jasu. Nevýhodou této metody je to, že porušuje tenké linie a ostré rohy, v případě filtrace na obdélníkovém okolí. Použijeme-li okolí např. ve tvaru kříže, nebo úsečky, je tento nedostatek eliminován.



Obr. 1.3 – Jiné tvary okolí pro filtraci mediánem

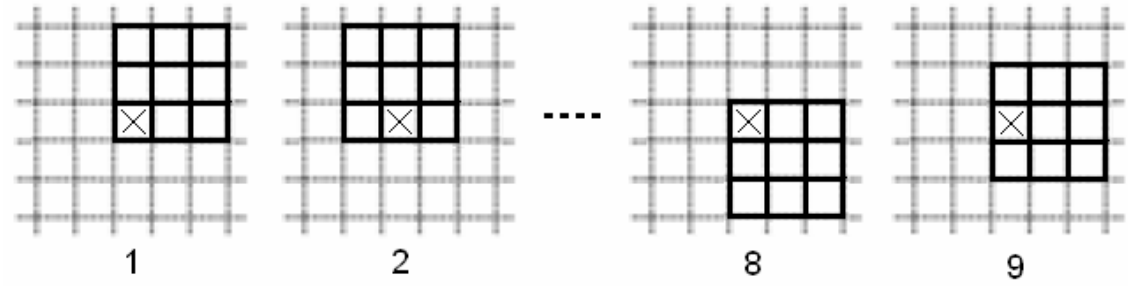
Další metodou je **Gaussův filtr**. Jedná se o metodu, při které se hodnoty jasu jednotlivých pixelů vypočtou diskrétní konvolucí s maskou, která obsahuje váhové koeficienty podle Gaussovy funkce:

$$G(i,j) = \exp\left(-\frac{i^2 + j^2}{2\sigma^2}\right). \tag{1.13}$$

Díky tomuto uspořádání masky mají koeficienty poblíž středu větší váhu než ty, co jsou na okraji. Nová hodnota jasu je pak dána vztahem:

$$f(x,y) = \sum_{i=-n}^{n} \sum_{j=-n}^{n} G(i,j) * g(x+i, y+j). \tag{1.14}$$

Při **filtraci rotující maskou** hledáme k filtrovanému bodu takové okolí, ke kterému bod patří. Příslušnost bodu k okolí posuzujeme podle kritéria homogenity, konkrétně podle rozptylu jasu. Většinou používáme okolí 3x3, které v okolí 5x5 vyhledávají homogenní část. Přípustných okolí ke každému bodu je vždy 9. Na (Obr. 1.4) je naznačen postup rotující masky, další varianta tvaru rotující masky je uvedena v [1].

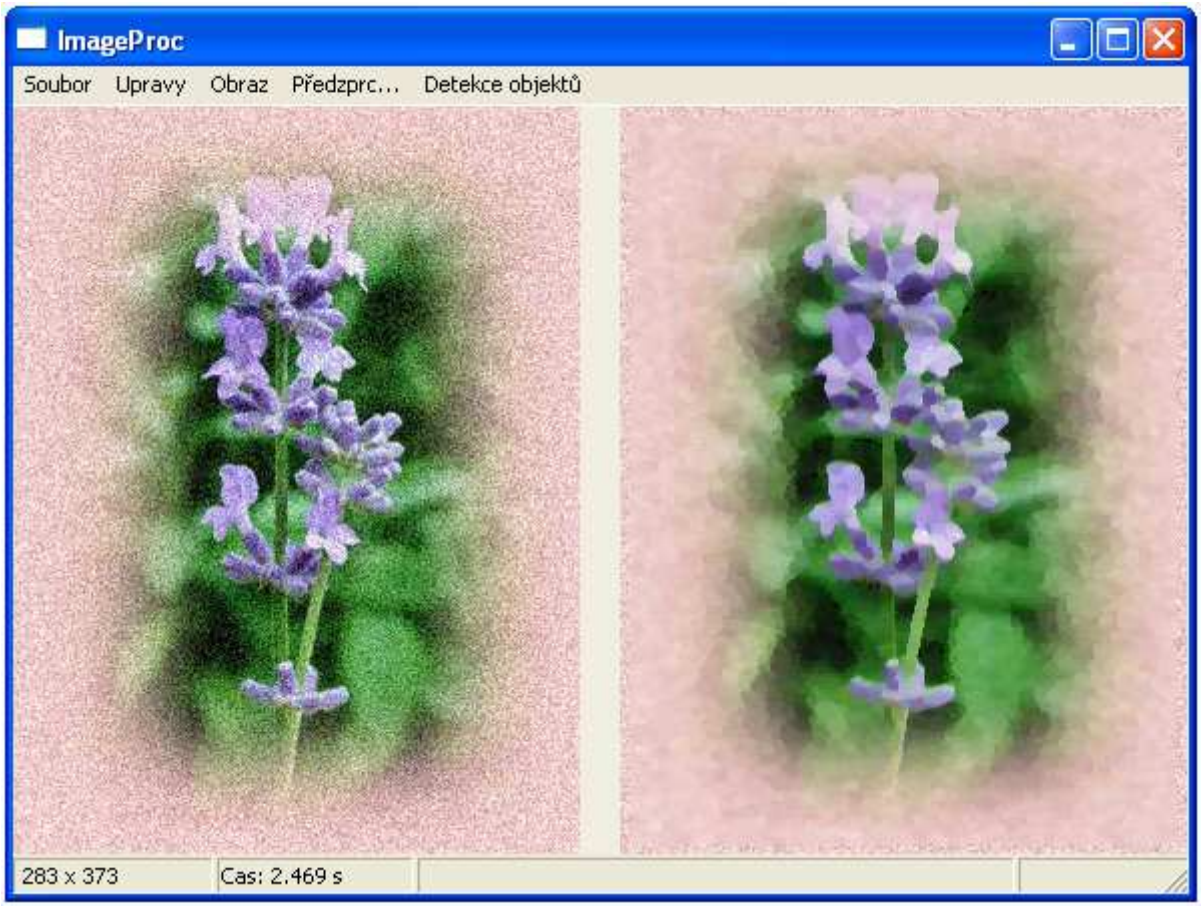


Obr.1.4 – Tvary rotujících masek

Rozptyl σ pro příslušná okolí vypočteme podle vzorce:

$$\sigma^2 = \frac{1}{9}\sum_{i=-n}^{n}\sum_{j=-n}^{n}[g(x+i, y+j) - \bar{g}(x, y)], \tag{1.15}$$

kde $\bar{g}(x, y)$ je průměrná hodnota jasu v daném okolí. Po té vybereme okolí s nejmenším rozptylem. Pro nalezení nové hodnoty jasu použijeme prostý průměr nebo medián z nalezeného okolí. Použití mediánu má význam hlavně při práci s binárním obrazem, protože při aplikaci průměru bychom získávali nepřípustné hodnoty jasu. Vlastností této metody je mírně ostřící charakter, také nedochází k rozmazávání hran. Nevýhodou je pak vysoká výpočetní náročnost, která rychle narůstá se zvětšujícím se okolí. Metoda rotující masky se dá použít v iteracích. Zpracování rychle konverguje ke stabilnímu stavu, kdy už se výsledný obraz nemění. Čím větší maska, tím méně iterací je třeba provést.



Obr. 1.5 – Filtrace rotující maskou na okolí 5x5

1.4.2 Hranové operátory a ostření obrazu

Hrana je vlastnost obrazového bodu a jako vektorová veličina je určena velikostí a směrem, viz. [2]. Vychází z gradientu jasové funkce. V místě hrany dochází k výrazným změnám obrazové funkce.

Velikost gradientu je tedy dána vztahem:

$$|\nabla g(X,Y)| = \sqrt{\left(\frac{\partial g}{\partial X}\right)^2 + \left(\frac{\partial g}{\partial Y}\right)^2}\,. \tag{1.16}$$

Gradient ukazuje ve směru největšího růstu obrazové funkce, který je dán vztahem:

$$\Psi = \tan^{-1}\left(\frac{\frac{\partial g}{\partial Y}}{\frac{\partial g}{\partial X}}\right), \quad \text{pro } \frac{\partial g}{\partial X} \neq 0\,. \tag{1.17}$$

Hranové operátory I

První skupinu hranových operátorů tvoří takové, které aproximují parciální derivaci obrazové funkce diferencí ve směru osy x a y:

$$|\nabla g(x,y)| = \sqrt{(\Delta_x g(x,y))^2 + (\Delta_x g(x,y))^2}\,, \tag{1.18}$$

kde

$$\begin{aligned} \Delta_x g(x,y) &= g(x,y) - g(x-1,y) && \text{je diference ve směru osy } x \\ \Delta_y g(x,y) &= g(x,y) - g(x,y-1) && \text{je diference ve směru osy } y. \end{aligned} \tag{1.19}$$

Diference pak prakticky realizujeme pomocí diskrétní konvoluce. Hlavní nevýhodou těchto operátorů je velká závislost na šumu.

Asi nejednoduší metodou nalezení obrazu hran je použití **Robertsova operátoru**. Používá ke svému výpočtu čtyř jeho nejbližších sousedů a tudíž je dost citlivý na šum. Velikost gradientu pak je dána vztahem:

$$|\nabla g(x,y)| = |g(x,y) - g(x+1,y+1)| + |g(x,y+1) - g(x+1,y)|\,. \tag{1.20}$$

Dalším operátorem pomocí něhož můžeme nalézt obraz hran, nikoliv však jejich směry, je **Laplaceův** operátor. Aproximuje druhou derivaci obrazové funkce a je invariantní vůči rotaci. Ve spojité oblasti je dán vztahem:

$$\nabla^2 g(X,Y) = \frac{\partial^2 g(X,Y)}{\partial X^2} + \frac{\partial^2 g(X,Y)}{\partial Y^2}, \tag{1.21}$$

v digitalizovaném obraze nahradíme parciální derivace diferencemi, nebo můžeme k výpočtu použít diskrétní konvoluci s maskou pro 4-sousedství nebo 8-sousedství. Jako výraznou nevýhodou lze uvést velkou citlivost na šum. Další varianty masek viz. [2].

$$h_4 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad h_8 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Potřebujeme-li získat obraz hran jen v některých směrech, použijeme **Sobelův operátor**, který aproximuje první parciální derivace a proto je také směrově závislý. Pro vstupní obraz provedeme konvoluci s maskami, přičemž použijeme masku otočenou jen do těch směrů, ve kterých chceme získat obraz hran.

$$h_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, h_2 = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}, h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, h_4 = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix},$$

další masky vzniknou postupnou rotací.

Obr. 1.6 – Obraz hran nalezen pomocí Sobelova operátoru

Další možností je, že, při stejném postupu jako u Sobelova operátoru použijeme jiné koeficienty v konvolučních maskách.

Mnohdy můžeme dosáhnout lepších výsledků s operátory:

Robinsonův: $h_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{bmatrix}$, **Kirschův**: $h_1 = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & 3 \\ -5 & -5 & -5 \end{bmatrix}$,

Prewittové: $h_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$.

Masky pro ostatní směry vzniknou opět rotací.

Pomocí gradientních operátorů docílíme také **zostření obrazu**. Ostření obrazu spočívá ve zvýšení strmosti hran. Pracujeme-li ve frekvenční oblasti, ostření odpovídá zdůraznění vysokých frekvencí. Výsledný obraz získáme podle vztahu:

$$f(x,y) = g(x,y) - C \cdot S(x,y), \tag{1.22}$$

kde C je kladný ostřící koeficient a $S(x,y)$ je obraz velikostí gradientů obrazové funkce.

Hranové operátory II

Druhá skupina hranových operátorů vychází z Marrovy teorie a detekuje hrany v místech, kde druhá derivace prochází nulou.

LoG operátor (Laplacian of Gaussian) jehož základním principem je hledání míst, kde druhá derivace obrazové funkce prochází nulou. Na tomto místě dosahuje první derivace lokálního maxima. Problémem je poměrně vysoká citlivost druhé derivace na šum, a proto se nejdříve potlačí šum a teprve pak se derivuje. Na potlačení šumu se většinou používá lineární výše zmíněný Gaussův filtr, jehož koeficienty v konvoluční masce odpovídají dvojrozměrnému gaussovskému rozdělení. Na odhad druhé derivace z filtrované obrazové funkce použijeme Laplacian ∇^2. Dostaneme tak LoG operátor, který lze díky linearitě upravit jako:

$$\nabla^2\left(G(x,y,\sigma) * g(x,y)\right) = \left(\nabla^2 G(x,y,\sigma)\right) * g(x,y). \tag{1.23}$$

Derivace Gaussova filtru $\nabla^2 G$ se dá vypočítat předem analyticky, protože nijak s obrazem nesouvisí, a tím lze snížit náročnost výpočtů. Po zderivování tedy dostaneme výsledný vztah pro výpočet konvoluční masky jako:

$$h(i,j) = c\left(\frac{i^2 + j^2 - \sigma^2}{\sigma^4}\right) \cdot e^{-\frac{i^2+j^2}{2\sigma^2}}, \tag{1.24}$$

kde c je normalizační koeficient.

Následné prahování LoG obrazu v intervalu hodnot blízkých 0 by vedlo k nalezení pouze slabých a nespojitých hran. Lepšího výsledku dosáhneme tehdy, použijeme-li masku 2x2 s reprezentativním bodem v levém horním rohu. Pokud dojde uvnitř okna ke změně znaménka, je nalezena skutečná hrana.

Při implementaci lze LoG operátor aproximovat tzv. DoG operátorem (Difference of Gaussians). Což je rozdíl dvou obrazů, se kterými byla provedena konvoluce s Gaussiánem, o různých σ.

Obr. 1.7 – Detekce hran pomocí LoG operátoru, $\sigma_1 = 1,1$ a $\sigma_2 = 0,9$

Hlavní nevýhodou této metody je pak to, že dochází k vyhlazování nebo ztrácení ostrých rohů. Spojování hran do souvislých křivek dalším nedostatkem operátoru, který je třeba odstranit při dalších úpravách.

Cannyho hranový detektor, založen na principu hledání hran v různých rozlišeních a měl by splňovat tyto tři kritéria, viz. [2]:

- Detekční kritérium – detektor nesmí opomenout žádnou významnou hranu
- Lokalizační kritérium – rozdíl mezi skutečnou a nalezenou hranou musí být minimální
- Detektor nesmí reagovat na jednu hranu vícenásobně

Cannyho detektor využívá konvoluci obrázku s dvojrozměrným Gaussiánem a derivaci ve směru gradientu *n*. Poskytuje tedy informaci o směru a velikosti hrany.

Operátor $G_n = \frac{\partial G}{\partial n} = n\nabla G$, představuje první derivaci G ve směru n, kde G je 2D Gaussián podle rovnice (1.13). Směr gradientu není předem znám, ale je možné ho odhadnout podle:

$$n = \frac{\nabla(G * g(x,y))}{|\nabla(G * g(x,y))|}. \tag{1.25}$$

Hranu pak nalezneme v místě, kde funkce $G_n * g(x,y)$ dosáhne lokálního maxima a druhá derivace se rovná nule. Tedy:

$$\frac{\partial^2}{\partial n^2} G * g(x,y) = 0, \tag{1.26}$$

pomocí této rovnice lze nalézt lokální maxima ve směru kolmém na hranu. Tato operace se někdy nazývá potlačení odezev mimo maxima. Pro velikost hrany platí:

$$|G_n * g(x,y)| = |\nabla(G * g(x,y))|. \tag{1.27}$$

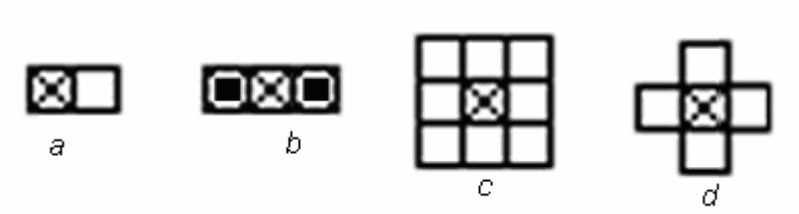
Prahováním s hysterezí jsou pak odstraněny vícenásobné odezvy a nesouvislost hran, které by měli tvořit souvislou hranici. Podle odhadovaného poměru signálu k šumu můžeme automaticky určit meze pro prahování.

Při detekci hran jsou vyzkoušena všechna měřítka a informace z nich se sdruží. Detektor může mít významnější odezvy na hranu ve více měřítkách, v tom případě je použit operátor v nejmenším měřítku, protože nejlépe lokalizuje hranu.U této metody se někdy také zavádí tzv. *syntéza odezev v různých měřítkách*. Z odezev detektoru pro nejmenší měřítko jsou syntetizovány hrany pro větší měřítka. Pak se syntetizované odezvy porovnávají se skutečnými pro příslušná větší měřítka σ. Skutečné hrany jsou do výsledku zařazeny tehdy, jsou-li silnější než syntetizované odezvy.

1.5 Matematická morfologie

1.5.1 Morfologické operace s binárními obrazy

Podle [2] nazýváme morfologickou operací relaci mezi obrazem a strukturním elementem, což je jiná, menší bodová množina. Při moroflogii pracujeme s obrazem i elementem jako s množinami uspořádaných dvojic. Každý strukturní element má svůj reprezentativní bod, který může či nemusí být jeho prvkem.



Obr. 1.7 – Příklady strukturních elementů (reprezentativní bod označen křížkem), b) speciální případ – reprezentativní bod není prvkem elementu (prvky elementu jsou zdůrazněny černými čtverečky)

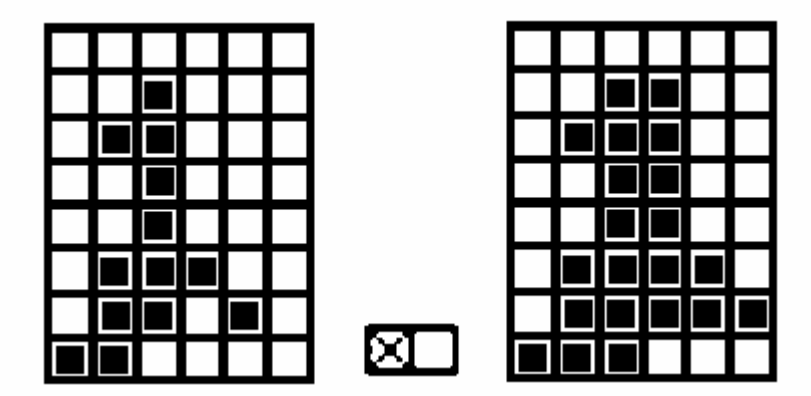
Morfologickou operaci realizujeme tak, že systematicky posouváme strukturní element po obraze. Mezi podmnožinou obrazu, kterou element právě pokrývá, a množinou elementu provedeme relaci. Výsledek pak zapíšeme do výstupního obrazu, a to na souřadnice odpovídající reprezentativnímu bodu ve vstupním obraze.

1.5.2 Dilatace

Dilatace je vektorový součet dvou množin, který můžeme definovat takto:

$$A \oplus B = \{p \in E_2 : p = a + b; a \in A, b \in B\}, \tag{1.28}$$

kde A je zpracovávaný obraz, B je strukturní element. Prakticky to znamená, že provedeme sjednocení A a B, všude tam, kde se aktuální bod rovná reprezentativnímu bodu elementu. Dilatací tedy docílíme zvětšení objektů na úkor pozadí a zacelení děr a zálivů.



Obr. 1.8 – Dilatace

Dilataci lze vyjádřit také jako sjednocení posunutých bodových množin, čili:

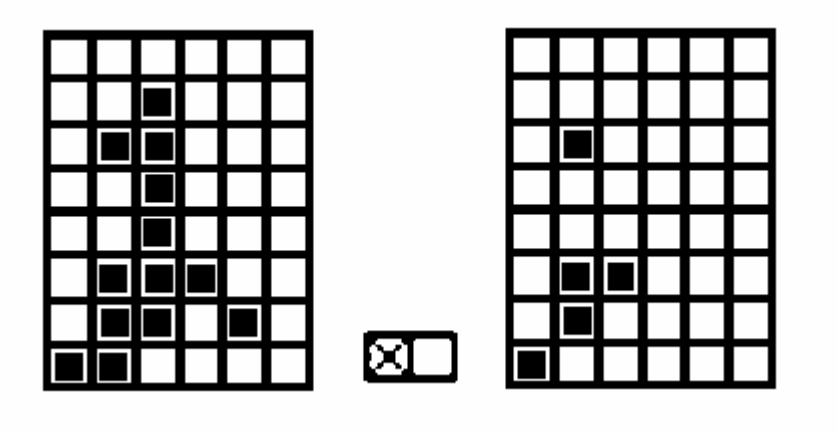
$$A \oplus B = \bigcup_{b \in B} A_b \ . \tag{1.29}$$

1.5.3 Eroze

Eroze je duální operace k dilataci, nikoliv však inverzní. Můžeme ji definovat takto:

$$A\Theta B = \{p \in E_2 : p + b \in A, \forall b \in B\}. \tag{1.30}$$

V podstatě jde o porovnávání zda-li pro všechna možná $p+b$ leží výsledek v A. Je-li splněna tato podmínka, je na aktuální bod obrazu zapsána „1“, v opačném případě „0“. Při erozi tedy dochází ke ztenčení objektů, ořezání úzkých výčnělků, rozdělení objektů na více částí, popřípadě zmizení objektů o stejné tloušťce jako strukturní element.



Obr. 1.9 – Eroze

Pomocí eroze lze také snadno nalézt obrysy objektů. Stačí když od obrazu odečteme jeho erozi. Abychom dostali celý obrys, musíme použít izotopický strukturní element (stejné vlastnosti ve všech směrech).

Erozi je možné také vyjádřit jako průnik všech posunů obrazu A o vektory $-b \in B$, čili:

$$A\Theta B = \bigcap_{b \in B} A_{-b}\ . \tag{1.31}$$

Obr. 1.10 – Odečtení erodovaného obrazu od původního

1.5.4 Otevření

Provedeme-li erozi a následně dilataci se stejným strukturním elementem získáme novou operaci tzv. otevření. Při použití této operace docílíme rozdělení objektů, které jsou spojeny tenkými liniemi, a vyhlazení hranice. Celkový tvar objektu se však příliš nezmění. Otevření je definováno takto:

$$A \circ B = (A \Theta B) \oplus B \,. \tag{1.32}$$

1.5.5 Uzavření

Duální operací k otevření je uzavření, což je dilatace následovaná erozí se stejným strukturním elementem. Uzavření využijeme v případě, že potřebujeme zaplnit úzké díry a zálivy, nebo také spojit objekty, které jsou blízko sebe. Velikost děr, které budou zaplněny pochopitelně závisí na velikosti strukturního elementu.

$$A \bullet B = (A \oplus B) \Theta B \,. \tag{1.33}$$

1.5.6 Tref či miň (Hit or miss)

Jedná se o transformaci, která podle [2] sleduje shodu mezi strukturním elementem a částí obrazu. Pomocí této metody jsme schopni nalézt rohy, hranice objektů, nebo objekty ztenčovat. Používá se složený strukturní element $B = \{B_1, B_2\}$, kde B_1 je podmnožinou množiny A (objekty v obraze) a B_2 je podmnožinou A^C (pozadí).

Do výsledného obrazu je zapsána „1“, jsou-li splněny následující podmínky pro aktuální bod *a*: Za předpokladu, že reprezentativní bod elementu B_1 je na souřadnicích bodu a, tak prvek B_1 je obsažen v A a zároveň prvek B_2 není obsažen v A^C. Transformaci tref či miň pak realizujeme pomocí eroze a dilatace:

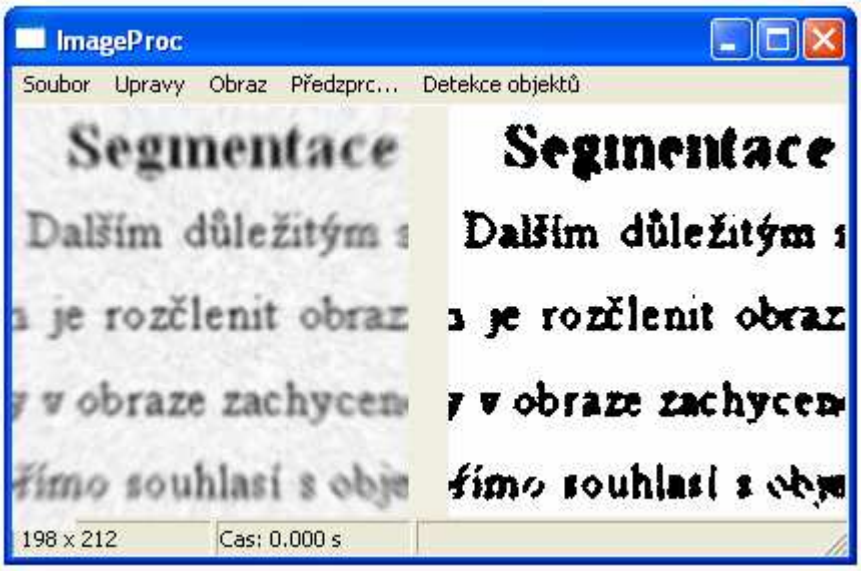
$$A \otimes B = \left(A \Theta B_1\right) \cap \left(A^C \Theta B_2\right) = \left(A \Theta B_1\right) \setminus \left(A \oplus \breve{B}_2\right). \tag{1.34}$$

2. Segmentace obrazu

Dalším důležitým stupněm při zpracovávání obrazů je segmentace. Prvořadým úkolem je rozčlenit obraz na pokud možno nesouvislé oblasti, které nějak souvisí s objekty v obraze zachycené. Kompletní segmentací rozumíme nalezení takových oblastí které přímo souhlasí s objekty ve vstupním obraze. K tomu je však zapotřebí spolupráce s vyšší úrovní zpracování obrazu, což není předmětem této práce, a proto se spokojíme se segmentací částečnou. Vytvořené segmenty jsou tedy jen homogenní z hlediska určitých zvolených vlastností, jako např. jas, barva apod. Výsledkem jsou pak většinou seznamy nalezených oblastí, rovnice hraničních křivek, řetězové kódy nebo jen upravená obrazová data pro další zpracování. Významným přínosem je také značné snížení objemu zpracovávaných obrazových dat.

2.1 Prahování

Jedná se o základní metodu segmentace. Důležitým předpokladem je to, že homogenní jas objektů a pozadí se dostatečně liší. Princip prahování spočívá v tom, že bodům s hodnotou jasu menší než zvolený práh přiřadíme hodnotu 0, bodům s hodnotou jasu větší než práh přiřadíme hodnotu 255. Výsledkem prahování je tedy binární obraz. Prahování je pro svoji jednoduchost a nenáročnou softwarovou realizovatelnost často používanou metodou, a lze ji provádět v reálném čase.



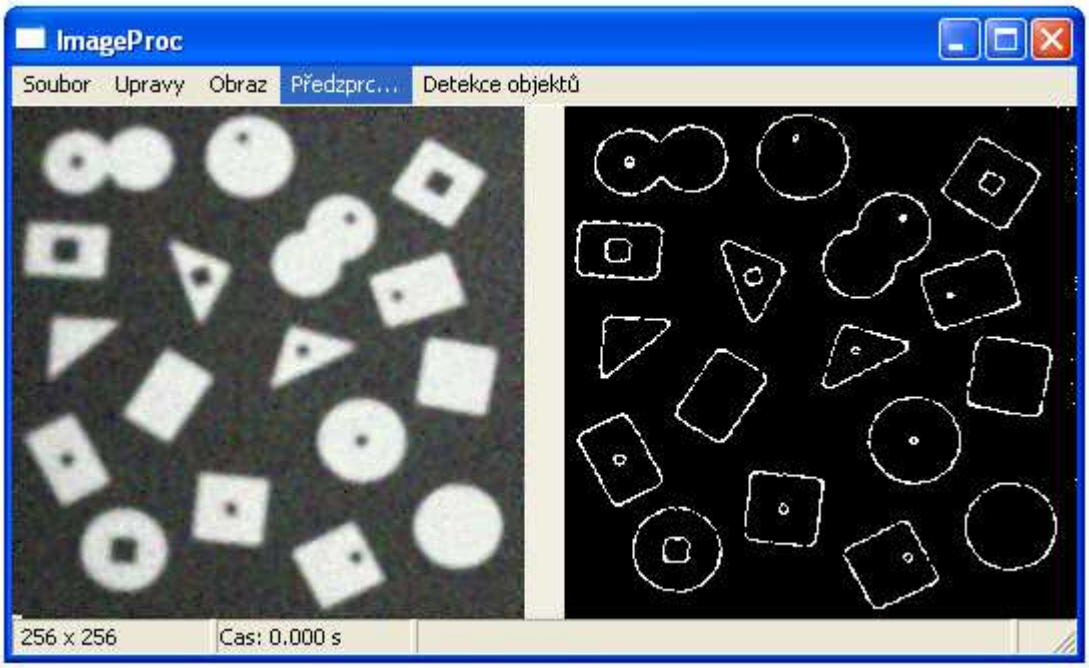
Obr. 2.1 – Procentní prahování s prahem 17%

Nevýhodou této metody je automatická volba prahu. Automaticky je možné zvolit práh jako hodnotu mediánu či průměru histogramu, ale rozumný výsledek nemusí být vždy zaručen. Proto se práh často volí interaktivě, nebo na základě histogramu. Také je možné určit tzv. procentní práh (Obr. 2.1). Práh se určí tak, aby po prahování

objekty pokrývaly stanovené procento původní plochy. Výpočet je možné provést z histogramu. Tato metoda volby prahu je vhodná zejména pro úpravu textu.

V některých případech je jas objektů i pozadí tak proměnlivý, že při prahování s jedním prahem nedosáhneme požadovaných výsledků. Tato proměnlivost může být způsobena například nestejnoměrným osvětlením. Řešením je použití proměnného (lokálního) prahu, jehož hodnota je určována na základě lokálních vlastností obrazu.

Další modifikací je prahování v určitém intervalu jasů. Za předpokladu, že hraniční body objektů nabývají jasových hodnot v určitému intervalu a zároveň se tyto hodnoty nevyskytují jinde v obraze, jsme schopni tímto způsobem jednoduše nalézt hranice objektů.



Obr. 2.2 – Prahování v intervalu <100, 150>

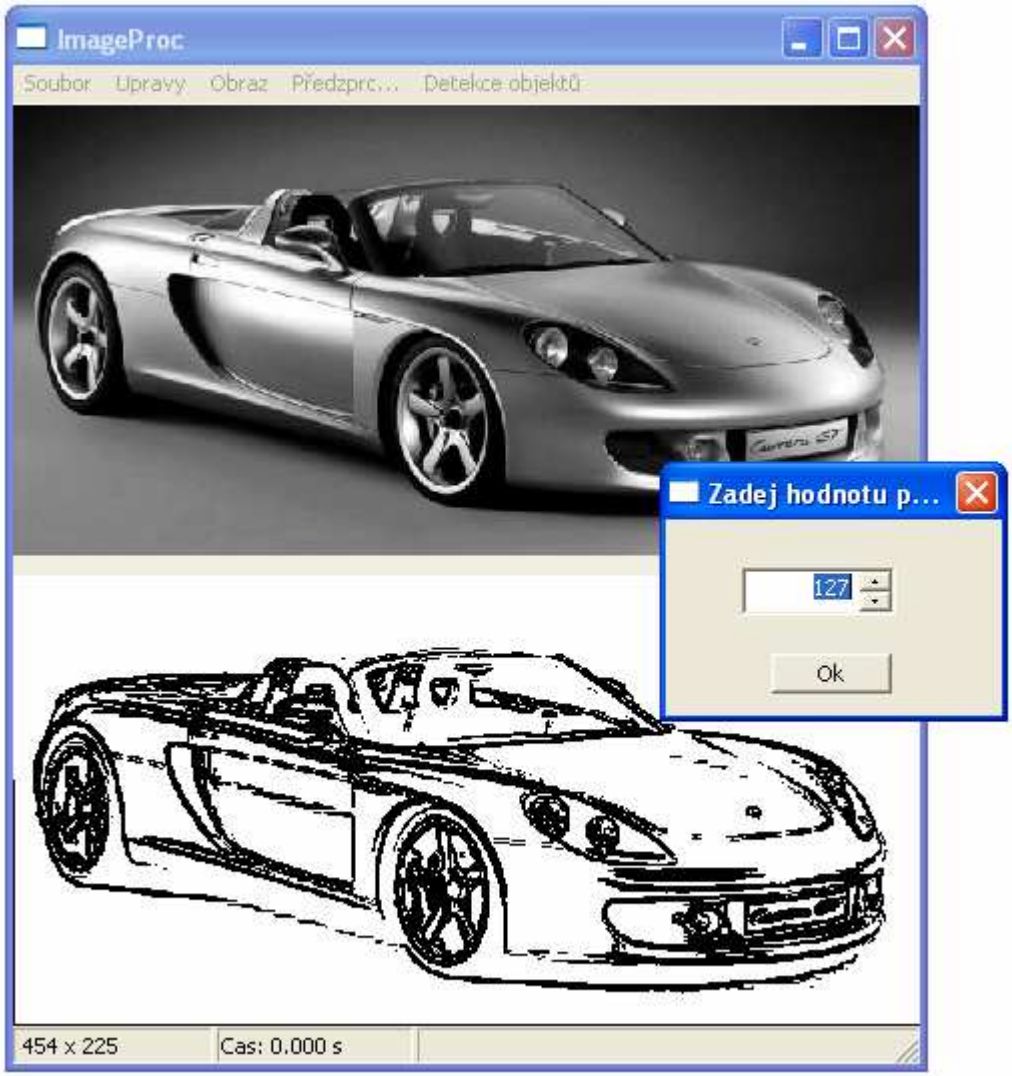
Prahování s více prahy je další metodou, kdy vznikne obraz obsahující několik jasových úrovní. Speciálním případem této modifikace je poloprahování, které umožní zachovat informaci o rozložení jasu v objektech, zatímco pozadí je odstraněno.

Zřejmá je i možnost prahovat barevný obraz s různými prahy v jednotlivých barevných kanálech.

2.1.1 Prahování obrazu hran

Většinou předpokládáme, že hranice objektů v obraze jsou tvořeny hranami, tedy místy s výraznými změnami jasové funkce. Obraz hran získáme pomocí hranových operátorů zmíněných výše. Ten však stále obsahuje spoustu nadbytečné obrazové informace v podobě hran, které neodpovídají skutečným hranicím hledaných útvarů.

Prahováním s vhodně zvoleným prahem tyto nevýznamné hrany odstraníme. Ve výsledném obraze pak zůstane tím méně důležitých hran čím bude práh vyšší. Tato metoda dosahuje dobrých výsledků zejména u obrazových dat s bez šumu.



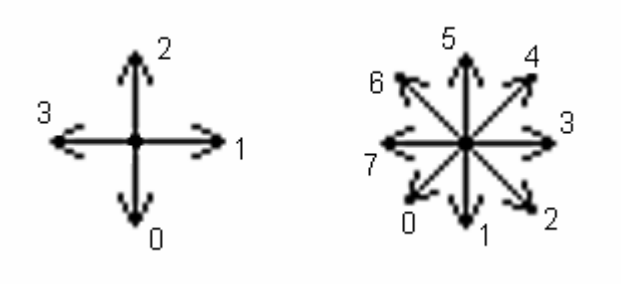
Obr. 2.3 – Prahování obrazu hran (hrany nalezeny pomocí Sobelova operátoru)

2.2 Sledování hranice

Cílem této metody je nalezení polohy a tvaru hranice všech objektů v obraze. Předpokládáme, že vstupní obraz je binární, popřípadě mohou být oblasti odděleny barvením.

Obraz je systematicky procházen po řádcích, dokud není nalezen první bod objektu, popř. díry. Pak je zkoumáno okolí tohoto bodu a to v protisměru hodinových ručiček, podle schématu (Obr. 2.3), dokud není nalezen další hraniční element. Do proměnné je uložen kód odpovídající směru postupu od předchozího bodu. Tímto způsobem projedeme celou hranici objektu dokud není nalezen opět první element.

Takto získáme řetězový kód reprezentující vnitřní hranici objetu. Algoritmus je možné modifikovat tak, aby našel také vnější hranice objektů. Podrobněji viz. [1].



Obr. 2.4 – Schéma směr postupu pro 4 – sousednost a 8 – sousednost

2.3 Houghova transformace

Houghova transformace je využitelná v případech, máme-li k dispozici určitou informaci o tvarech objektů, které chceme vysegmentovat. Touto informací máme namysli druh křivek, ze kterých je hranice složena. Pak lze chápat úlohu segmentace spíše jako nalezení daných objektů v obraze. Klasická Houghova transformace je použitelná hlavně pro detekci přímek, kružnic a elips, protože jejich parametrický popis je jednoduchý. Existuje samozřejmě také zobecnění, určené pro složitější tvary hranic. Princip metody je dobře vidět na úloze hledání přímek. Obecná rovnice přímky má tvar:

$$x_2 = kx_1 + q\,. \tag{2.1}$$

Jako v [7] použijeme pro transformaci normálový tvar:

$$r = x_1 \cos(\varphi) + x_2 \sin(\varphi), \tag{2.2}$$

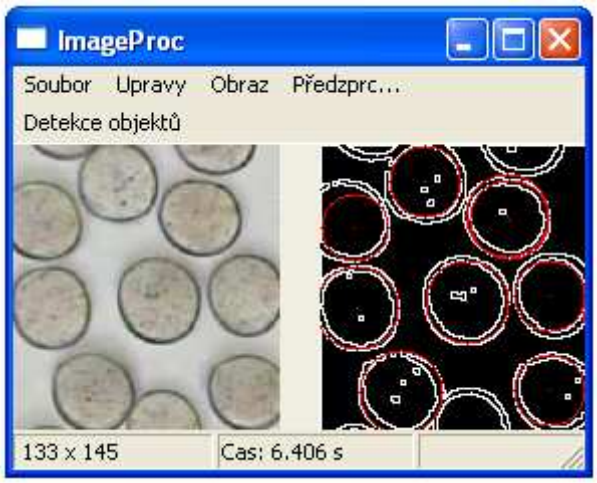
protože proměnné r a φ nabývají konečných hodnot, tedy $\varphi \in \langle 0,360 \rangle$ a r je omezeno velikostí obrázku. Nejdříve si nadefinujeme tzv. akumulátor. V tomto případě se jedná o 2-rozměrné, vynulované pole celočíselného typu, jehož velikost je dána počtem přípustných hodnot proměnných r a φ. Pak systematicky procházíme obrázek (předpokládaným vstupem je binární obraz hran) dokud není nalezen bod objektu. Souřadnice nalezeného bodu použijeme jako parametry x_1 a x_2 do rovnice (2.2) a postupně dosazujeme všechny přípustné hodnoty proměnné φ. Takto vždy získáme dvojici $[r,\varphi]$, což jsou souřadnice položky v akumulátoru, kterou inkrementujeme. Tento postup aplikujeme na celý obraz.

Dalším krokem je nalezení lokálních maxim v akumulátoru. Ty totiž odpovídají přímkám obsaženým v obraze. Maxima mohou dosahovat různě velkých hodnot. To je dáno odlišnou délkou hledaných úseček, popřípadě množstvím bodů, které k úsečce

patří. Výsledkem Houghovy transformace je pak soubor dvojic parametrů $[r,\varphi]$, pomocí nichž snadno získáme rovnice přímek vyskytujících se v obraze.

Analogickým postupem lze aplikovat tuto metodu pro detekci kružnic, které jsou dány rovnicí: $(x_1 - a)^2 + (x_2 - b)^2 = r^2$. Z rovnice je patrné, že tentokrát budeme potřebovat 3-rozměrný akumulátor, čímž značně vzroste výpočetní náročnost.

Hlavní výhodou této metody je značná necitlivost vůči šumu a určitá tolerance odchylek hledaných tvarů oproti jejich analytickému popisu.



Obr. 2.5 – Kružnice nalezené Houghovou transformací

2.4 Narůstání oblastí

Cílem těchto segmentačních metod je rozdělení obrazu na maximální oblasti, splňující nějaké kritérium homogenity. Kritériem může být např. hodnota jasu v dané oblasti, či textura.

Základní metodou je **spojování oblastí** na základě jasu. Předpokládejme, že na začátku každý pixel představuje jednu oblast. Obraz systematicky procházíme a vždy testujeme sousední oblasti zda splňují kritérium homogenity. V případě, že ano, je z oblastí vytvořena nová, které je přiřazena nová hodnota jasu, vypočtená např. průměrem. Algoritmus končí tehdy, nejsou-li v obraze žádné další sousední oblasti ke spojení. Výsledná podoba vytvořených oblastí pak závisí na tom, ze kterého místa v obraze vycházíme. Důsledkem některých spojení se někdy změní jas oblasti natolik, že ji již není možné spojit s jinou, se kterou by byla spojena, kdybychom začali např. v protějším rohu.

Duálním postupem je **štěpení oblastí**. Na počátku je z obrazu vytvořena jedna oblast, která však nesplňuje kritérium homogenity, a proto je rozdělena na menší. Oblasti jsou dále děleny až do té doby, dokud nesplňují dané kritérium. Přestože je

princip obrácený jako u spojování oblastí, nedosahujeme stejných výsledků, i když použijeme stejné kritérium. Další modifikace narůstání oblastí viz. [1].

3. Dynamicky linkované knihovny v prostředí Microsoft Visual C++

3.1 Obecně

DLL knihovna je samostatný programový celek obsahující proměnné, funkce, procedury, zdroje, či třídy, které pak poskytuje ostatním aplikacím. Jedná se tedy o jednotný spustitelný kód, umístěný ve vlastním souboru (*.dll). Díky tomu je možné psát rozsáhlé aplikace modulově. Hlavní výhodou je to, že provedeme-li nějaké změny, nemusíme znovu kompilovat celou aplikaci. Při distribuci pak stačí rozšířit příslušnou modifikovanou knihovnu.

Knihovna poskytuje aplikační rozhraní, prostřednictvím něhož jsou exportovány funkce a proměnné, klientské aplikaci. Aby bylo možno tyto funkce použít je nutné knihovnu načíst do virtuálního prostoru aplikace. Z knihovny exportujeme pomocí direktivy:

extern "C" __declspec(dllexport) TYP JmenoFunkce(Parametry);.

V klientské aplikaci pak získáme adresu exportované funkce pomocí funkce *FARPROC GetProcAddress(HMODULE hModule, LPCSTR lpProcName);* a nebo můžeme funkci z knihovny importovat pomocí:

extern "C" __declspec(dllimport) TYP JmenoFunkce(Parametry);..

Každý soubor *.dll musí obsahovat funkci *DllMain*, která je ihned po načtení knihovny volána funkcí *_DllMainCRTStarup* tzv. vstupním bodem knihovny. Funkce *DllMain* je volána také při uvolnění knihovny z virtuálního prostoru klientské aplikace. Podrobněji viz. [5].

3.2 Dynamické načtení (linkování)

Výhodou tohoto způsobu je možnost načíst knihovnu až ve chvíli, kdy je volána některá z exportovaných funkcí. Stejně tak je možné knihovnu uvolnit v případě, kdy již její funkce nebudeme potřebovat. Při spuštění klienta není kontrolována existence souboru *.dll, díky tomu můžeme sami za běhu programu rozhodnout jakou verzi knihovny načíst, popřípadě dát uživateli možnost volby.

3.3 Statické načtení (linkování)

Při statickém linkování je kromě souboru *.dll ještě vytvořen ještě soubor *.lib (tzv. importovací knihovna), který představuje zástupce knihovny a obsahuje všechna exportovaná symbolická jména (identifikátory pro exportované funce a třídy). Tato importovací knihovna musí být přidána do projektu klientské aplikace. Při sestavování

klienta jsou importované symboly porovnávány s exportovanými symbolickými jmény souboru *.lib. Při spuštění aplikace je automaticky kontrolována existence souboru *.dll, bez ohledu na to, zda je některá importovaná funkce použita.

4. Dokumentace implementovaných algoritmů a testovací aplikace

Cílem zadání bylo realizovat dynamicky linkovanou knihovnu, která bude obsahovat algoritmy pro předzpracování obrazu.

4.1 Rozhraní knihovny PImage

Jako rozhraní dll knihovny byl vytvořen soubor *interface.h*, jehož kompletní obsah je uveden v příloze č.2. Základním kamenem je struktura *CImage*, která reprezentuje právě načtený obrázek. Jedním ze základních požadavků zadání byla použitelnost knihovny v jiných programovacích jazycích a prostředích. Z tohoto důvodu by rozhraní mělo být jednoduché a hlanvě neobjektové. Proto také CImage neobsahuje objekty ani metody.

```
typedef struct _TCImage
{
	PBYTE Data8, m_pPxs;
	_TPixels32 *Data32;
	RGBQUAD *pTab;
	unsigned char Priznak;
	BITMAPINFOHEADER InfoHead;
	BITMAPFILEHEADER FileHead;
	int *histogram;
}CImage;
```

Data8, *m_pPxs* a *Data32* jsou pointery odkazující na počátky obrazových dat, ať už se jedná o šedotónový obrázek nebo barevný, kde jeden pixel je uložen jako čtyřbytová sekvence. Pro usnadnění přístupu k jednotlivým barevným složkám jsem nadefinoval datový typ *TPixels32*. Parametr *pTab* je ukazatelem odkazujícím na tabulku použitých barev v obrázku. Používá se v případech, je-li tabulka v obrázku použita, např. jedná-li se o obraz s bitovou hloubkou 8bpp. Proměnná *Priznak* je určena pro nastavení různých atributů charakterizující zpracovávaný obrázek. *Priznak* obsahuje zatím 4 atributy, které jsou uloženy na dolních čtyřech bitech, ostatní jsou rezervovány pro další použití. V tabulce (1) jsou uvedeny významy jednotlivých atributů. Nastavování a testování hodnot *Priznak* je realizováno pomocí maker definovaných v knihovně. Tato makra však už nejsou součástí rozhraní knihovny. Struktury *FileHead* a *InfoHead* slouží pro ukládání dalších vlastností bitmapy, jako např. rozměry, velikost souboru atd. Tyto struktury jsou standardně definovány v C++.

Bit	Význam
0.	0 – jedná se o bitmapu
	1 – jedná se o jpeg
1.	0 – obrazová data jsou ve formátu YUV
	1 – obrazová data jsou ve formátu RGB
2.	0 – barevný obrázek
	1 – obrázek v odstínech šedi
3.	0 – obraz není binární
	1 – binární obraz

Tabulka 1 – Významy bitů proměnné příznak

Pro použití knihovny se předpokládá, že v externí aplikaci je vytvořena alespoň jedna instance třídy *CImage*, se kterou pracují exportované funkce.

4.2 Exportované funkce

Exportované funkce, které jsou implementovány v knihovně by se daly rozdělit do dvou částí. Do první patří funkce vstupní/výstupní, které jsou určeny pro načtení, zobrazení a uložení obrazových dat, do druhé pak funkce pro vlastní zpracování obrazu.

4.2.1 Vstupní a výstupní funkce

BOOL LoadFromFile(char *szJmeno, CImage *pDest)

Funkce slouží pro načtení obrázku ze souboru. V parametru *szJmeno* se předává jméno souboru, ze kterého chceme obraz načíst. Ve druhém se pak předává ukazatel na strukturu *CImage*, do které chceme data načíst. Funkce vyhodnotí zda se jedná o bitmapu nebo o jpeg, v závislosti na tom pak budou volány příslušné lokální funkce, které provedou načtení dat a vyplnění hlaviček bitmapy. Tyto budou vysvětleny dále. Pak jsou ještě volány funkce pro alokaci a zkopírování obrazových dat do pomocných instancí třídy *CImage*, které slouží např. pro obnovení původního obrazu, aniž bychom ho znovu načítat ze souboru. Je-li obraz korektně načten funkce vrátí hodnotu TRUE, v opačném případě FALSE. Důvodem neúspěchu funkce *LoadFromFile* může být nepodporovaný grafický formát, nečitelnost souboru a pod.

BOOL SaveToFile(char *szJmeno, CImage *pSrc, int nQuality)

Když budeme chtít obrázek uložit, tak použijeme tuto funkci. V prvním parametru musíme předat jméno výstupního souboru, dále pak ukazatel typu *CImage*,

kde jsou uložena zdrojová data. Má-li být výstupní soubor bude ve formátu jpeg, je nutné ve třetím parametru zadat ještě míru komprese(v případě bitmapy předáváme hodnotu -1). Pak je volána lokální funkce *SaveJpg* pro kódování jpegu a zapsání do souboru. Jedná-li se o bitmapu, testuje se zda nemá být výstupní obrázek v odstínech šedi, v případě že ano vygeneruje se tabulka barev, pak je zapsána hlavička i obrazová data. V případě barevného obrázku stačí jen zaspat hlavičku a obrazová data. Proběhne-li zapsání všech částí souboru v pořádku funkce vrací hodnotu TRUE. V opačném případě je vráceno FALSE.

void Vykresli(HDC hDC,HWND hMuj, CImage *Src)

Pro zobrazení obrázku, a to jak upravovaného tak i originálu je zde funkce *Vykresli*. Jako první parametr je předán handle kontextu zařízení, dále je třeba předat handle okna, do kterého se má obrázek vykreslit, a na konec ukazatel na strukturu *CImage*, která obsahuje všechny potřebné informace a zdrojová data. Funkce vrací hodnotu TRUE v případě, že vykreslení proběhne v pořádku.

BOOL CopyFromOrig(CImage *pDest, BOOL co)

Funkce zkopíruje obrazová data z jedné ze záložních instancí třídy *CImage*, která je nadefinována uvnitř knihovny, do vnější instance (ta se kterou pracují exportované funkce). To jestli se jedná pouze o krok zpět, či úplné obnovení originálu specifikujeme parametrem *co*. Funkce vrací hodnotu TRUE.

void Histogram(CImage *pSrc)

Funkce vypočte histogram z obrazu uloženého v *pSrc*. Pro barevné obrazy je vypočten z hodnot odpovídajících šedotónovému obrazu.

void Destroy(CImage *pSrc)

Funkce dealokuje a vynuluje veškerou paměť, která byla použita. Uvolní záložní instance třídy *CImage*.

BOOL Resample(CImage *pDest,INT nW, INT nH)

Funkce převzorkuje obraz na zadané nové rozměry, za pomoci bikubické interpolace. Druhým a třetím parametrem jsou předány nové rozměry obrazu pixelech.

4.2.2 Funkce předzpracování a segmentace

Všem funkcím pro úpravu obrazu je třeba jako první parametr předat ukazatel na strukturu *CImage*. Dále pak všechny funkce vrací hodnotu TRUE v případě úspěšného provedení operace.

BOOL DoCB(CImage *pSrc)

Funkce převede barevný obraz do odstínů šedi. Provedou se příslušné úpravy hlavičky. Nové hodnoty jasu jsou vypočteny s ohledem na citlivost lidského oka vnímat různé barvy různě.

BOOL Negativ(CImage *pSrc)

Funkce převede obraz do negativu

BOOL NormalizaceJasu(CImage *pSrc)

Funkce transformuje obraz tak, aby bylo využito všech 256 jasových úrovní. Jedná-li se o barevný obraz jsou obrazová data převedena z formátu RGB do YUV pomocí lokální funkce *RgbToYuv*. Pak se provede normalizace s jasovou složkou Y. Po normalizaci jsou data opět převedena do RGB a to kvůli zobrazení. U černobílých obrázků je normalizace provedena hned. Tohoto postupu je využito také u následujících transformací, kde se pracuje jen jasovou složkou.

BOOL EkvalizaceHist(CImage *pSrc)

Funkce provede vyrovnání (ekvalizaci) histogramu.

BOOL FiltrPrumer(CImage *pSrc, int n)

Funkce provede filtraci prostým průměrováním. Druhým parametrem je rozměr okolí filtrovaného bodu, $n = 1$ odpovídá okolí 3x3, $n = 2$ odpovídá okolí 5x5 atd. Krajní body obrazu nejsou do filtrace zahrnuty a to z důvodu urychlení výpočtu. Toto omezení je u všech operací, kde by jsme se mohli při výpočtech octnout za hranicí obrazu.

BOOL FiltrMed(CImage *pSrc, int n)

Funkce provede filtraci mediánem. Stejně jako v předchozím případě je potřeba zadat rozměr okolí. Medián je počítán ze čtvercového okolí. Pro seřazení hodnot jasů je použita funkce *qsort*, která je standardně implementována v C++.

BOOL FiltrRotMask(CImage *pSrc, int n, BOOL median)

Funkce provede filtraci rotující maskou. Stejně jako v předchozím případě je potřeba zadat rozměr okolí. Třetím parametrem je možné zvolit, zda se nová hodnota jasu filtrovaného bodu vypočte mediánem anebo průměrem. Kód je optimalizován pro často používané okolí 3x3.

BOOL FiltrGauss(CImage *pSrc, int n)

Funkce provede Gaussovu filtraci. Nejdříve se vypočte maska podle Gaussovy funce, pak je s touto maskou a okolím filtrovaného bodu provedena konvoluce.

BOOL HranyRoberts(CImage *pSrc)

Funkce vytvoří obraz hran pomocí Robertsova operátoru.

BOOL HranySobel(CImage *pSrc, int smer)

Funkce vytvoří obraz hran pomocí Sobelova operátoru. Druhým parametrem je možno určit v jakém směru budou hrany nalezeny. *Smer* může nabývat hodnot 0 až 8. Velikost hrany je vypočtena konvolucí okolí bodu s maskou v příslušném směru. Uživatel má i možnost zvolit pro výpočet všech osm směrů najednou, čemuž odpovídá *smer* = 8.

BOOL HranyLaplace(CImage *pSrc, int maska)

Funkce vytvoří obraz hran pomocí Laplaceova operátoru. Druhý parametr udává typ masky. Defaultně je nastaven typ h_4, nebo lze nastavit typ h_8 a to nastavením druhého parametru na hodnotu 8. Velikost hrany je pak počítána konvolucí s maskou.

BOOL HranyLoG(CImage *pSrc, double sigma1, double sigma2)

Funkce vytvoří obraz hran pomocí DoG operátoru, který je aproximací LoG operátoru. Prvním a druhým parametrem jsou hodnoty σ, pro výpočet konvolučních masek. Velikost hrany je počítána rozdílem dvou obrazových bodů o stejných souřadnicích, obrazy vznikly kovnolucí s maskami o různých σ. Výsledkem je binární obraz.

BOOL Prahovani(CImage *pSrc, int prah)

Funkce provede segmentaci prahováním. Jako druhý parametr je předána hodnota prahu, která může nabývat hodnot z rozsahu 0 až 255.

BOOL BarPrahovani(CImage *pSrc, int Rp, int Gp, int Bp)

Funkce provede prahování pro každý barevný kanál zvlášť. Hodnoty jednotlivých prahů jsou zadávány jako 2 až 4 parametr.

BOOL PrahovaniInt(CImage *pSrc, int dolni, int horni)

Funkce provede prahování v zadaném intervalu jasů. Bodům jejichž jas je uvnitř intervalu je přiřazena „1“, ostatním „0“.

BOOL ProcPrahovani(CImage *pSrc,int prah)

Funkce provede procentní prahování. Druhým parametrem je práh, který určuje, kolik procent plochy obrazu budou zaujímat objekty. Výpočet je proveden z histogramu.

BOOL Dilatace(CImage *pSrc)

Funkce provede dilataci obrazu s elementem, který vybere uživatel pomocí exportovaného dialogu z knihovny. Dilatace je realizována logickým součtem s maskou. Předpokládaným vstupem je pouze binární obraz, stejně tak jako u ostatních morfologických operací.

BOOL Eroze(CImage *pSrc)

Funkce provede erozi, tak že, pro každé přiložení masky testuje její shodu s obrázkem. Je-li shoda úplná výstupnímu bodu je přiřazena „1“.

BOOL NarOblasti(CImage *pSrc, int krit)

Funkce realizuje segmentaci narůstáním oblastí. Druhým parametrem se zadává kritérium homogenity, tedy o kolik jasových úrovní se mohou dvě oblasti lišit, aby ještě byly spojeny. Postupně je procházen celý obraz a vždy dvě sousední oblasti jsou spojeny, splňují-li kritérium. Postup je opakován do té doby, dokud je co spojovat.

BOOL SledHranice(CImage *pSrc, int s)

Funkce realizuje segmentaci pomocí algoritmu sledování hranice (podrobně popsán v kapitole 2.2. Druhým parametrem se volí zda se použije 4 – sousednost nebo 8 – sousednost. Předpokládaným vstupem je binární obraz. Výstupem je pak obraz s vyznačenými nalezenými objekty, resp. jejich hranice. Dále jsou objekty zaspány do souborů, kde je jejich pořadové číslo, souřadnice levého horního rohu a řetězový kód hranice, vytvořený podle schématu (Obr. 2.3).

BOOL HoughTrans(CImage *pSrc, int krivka,unsigned int minR, unsigned int maxR, int okoli, double faktor)

Funkce realizuje klasickou Houghovu transformaci (algoritmus podrobně popsán v kapitole 2.3). Zde už se jedná o pokročilejší algoritmus a proto jsem z časových důvodů implementoval jeho základní variantu pro detekci kružnic a přímek. U kružnic je detekce omezena pouze na ty, které leží alespoň z 80% procent v obraze. Předpokládaným vstupem je pouze binární obraz hran. Výstupem je pak obraz s vyznačenými detekovanými křivkami a soubor, který obsahuje parametry jednotlivých křivek. Druhým parametrem je určen druh křivek. *minR* a *maxR* jsou mezní hodnoty

poloměru hledaných kružnic. Parametr *okoli* určuje minimální vzdálenost dvou maxim v akumulátoru, doporučená hodnota je 5 až 30. Pomocí parametru *faktor* jsou oříznuta „nevýznamná“ maxima akumulátoru. Může nabývat hodnot z intervalu (0,1).

4.3 Lokální funkce

V knihovně byla definována řada dalších funkcí, které jsou využívány exportovanými funkcemi. Zde bych chtěl zmínit ty nejdůležitější.

`BOOL LoadBmp(char *szJmeno, CImage *pDest)`

Jedná-li se při načítání obrázku o bitmapu, je volána tato funkce. Funkci je třeba předat jméno souboru a ukazatel na objekt typu *CImage*, kam se uloží obrazová data. Nejdříve je na čtena hlavička (*BITMAPFILEHEADER* a *BITMAPINFOHEADER*). Načítání obrazových dat je rozděleno do čtyř větví, a to podle bitové hloubky. Knihovna podporuje tyto formáty: 32bpp, 24bpp, 8bpp, 1bpp. Každý formát je tedy zpracován jinak. Je-li obraz správně načten funkce vrátí hodnotu TRUE. Obrázek je převeden na bitmapu s bitovou hloubkou 32bpp a v tomto tvaru může být dále zpracováván.

`BOOL LoadJpeg(char *szJmeno, CImage *pDest)`

Jedná-li se při načítání obrázku o jpeg, je volána tato funkce. Funkci je třeba předat jméno souboru a ukazatel na objekt typu *CImage*, kam se uloží obrazová data. Pro dekódování jpegu byl použit volný software a to: Small JPEG Decoder Library v0.93b, Copyright (C) 1994-2000 Rich Geldreich. Jedná se o knihovnu pro dekódování i enkódování grafického formátu jpeg. Pro potřeby použití dekodéru byla nadefinována třída *CJpgInfo*, ve které jsou proměnné např. pro testování stavu, jak proběhla ta která část procesu dekódování.

Při dekódování jsou nejdříve provedeny nezbytné inicializace, týkající se vytvoření objektů *Pjpeg_decoder* a *Pjpeg_decoder_file_stream*, a získání rozměrů obrázku. Vlastní dekódování je pak prováděno po jednotlivých řádcích. Pro každý řádek je volána funkce *SetImageLine*, která zajistí správné uložení obrazových dat do naší struktury *CImage*. Proběhnou-li všechny části dekódování správně, jsou ještě nastaveny příslušné hodnoty do hlavičky bitmapy, nastaví se příslušné bity v proměnné *Priznak* a funkce se ukončí s návratovou hodnotou TRUE. Obrázek je nyní převeden na standardní bitmapu s bitovou hloubkou 32bpp.

`BOOL SaveJpg (CImage *pSrc, LPCTSTR szOutPath, INT nQuality)`

O zpětné enkódování obrazu do formátu jpeg je postará funkce *SaveJpg*. Jako první parametr je předán ukazatel na *CImage*, který odkazuje na zdrojový obraz. Jméno výstupního souboru včetně cesty je předáno ve druhém parametru, a dále se předává míra komprese. Pro enkódovaní stačí zavolat metodu `::JpgCompress`, která je součástí výše zmíněné knihovny, a předat jí tyto parametry: rozměry obrázku, bitovou hloubku, ukazatel na zdrojová data, jméno výstupního souboru a míru komprese. Funkce vrátí hodnotu TRUE v případě úspěšné kompresse.

`BOOL YUVtoRGB(CImage *pSrc), BOOL RGBtoYUV(CImage *pSrc)`

Funkce pro vzájemnou konverzi mezi barevnými modely. Po konverzi z RGB do YUV jsou data uložena na stejném místě (pSrc.Data32) stím, že složky YUV jsou uloženy ve složkách RGB v tomto pořadí. Obě funkce vrátí hodnotu TRUE.

4.4 Popis testovací aplikace

Jako předváděcí program byla vytvořena aplikace typu Win32 s názvem *ImageProc*. Jedná se o jednoduchou aplikaci sloužící výhradně pro demonstraci exportovaných funkcí knihovny PImage. Proto při její tvorbě nebyl kladen velký důraz na design či použitelnost v praxi. Aplikace je v podstatě tvořena jedním oknem a několika modálními dialogy. Uživatel má možnost volat jednotlivé funkce prostřednictvím hlavního menu a ihned sledovat co se děje s obrazem. Obraz je vykreslován dvakrát, s tím, že na druhém je možné vidět efekt příslušné operace. Pro otevření a uložení souborů jsem využil standardní systémové dialogy pro výběr souboru. Aplikace obsahuje také několik dialogů pro zadávání různých parametrů příslušným funkcím. Dialogy jsou načítány z prostředků a jsou určeny pro zadávání míry komprese JPEG, rozměru okolí při filtraci a pro volbu prahu a podobně. Parametry jako rozměry nebo dobu trvání nějaké operace lze sledovat na stavovém řádku.

Knihovna je dynamicky přilinkována při vytvoření hlavního okna aplikace a uvolněna při ukončení programu.

5. Porovnání s jinými dostupnými nástroji

V této kapitole bych se chtěl zmínit o dvou dalších open source knihovnách zabývajících se zpracováním obrazové informace. Mým cílem bylo porovnání knihoven resp. jejich algoritmů, z hlediska časové náročnosti.

5.1 FILTERS

Jedná se o knihovnu obsahující velké množství algoritmů pro zpracování obrazu a podporu počítačového vidění. Kromě standardních metod jako je např. konvoluce s různými typy masek, filtry, morfologické operace, hranové operátory, geometrické transformace obsahuje také nástroje pokročilejší jako např. Houghovu transformaci, Cannyho hranový detektor a další. Díky rozhraní, které je vytvořeno v ANSI C je knihovna použitelná v různých programovacích jazycích jako C, C++, VB, C#, Delphi, Java a také v běžných skriptovacích jazycích jako jsou Perl, Python, PHP, TCL nebo Ruby. Načítání a ukládání obrazu je realizováno pomocí externí knihovny FreeImage, pro některé další metody jsou použity knihovny OpenCV a Itk.

Testování implementovaných metod jsem provedl pomocí přiložené testovací aplikace – FiltersTest v3.0 (2007/01), která uživateli umožňuje otestovat všechny algoritmy obsažené v knihovně, sledovat dobu trvání jednotlivých metod a také ukládat upravené obrázky.

5.2 CxImage

CxImage je knihovna určená pro zpracování a převod obrazů mezi různými grafickými formáty. Obsahuje více jak 200 funkcí jako např. gama korekce, normalizace a ekvalizace histogramu, převzorkování, prahování, různé filtry, transformace mezi barevnými prostory adt. Podporovanými formáty jsou: BMP, GIF, ICO, CUR, JBG, JPG, JPC, JP2, PCX, PGX, PNG, PNM, RAS, TGA, TIF, WBMP, WMF. Může být linkována staticky, dynamicky a nebo na ActiveX. CxImage je možné bez problémů kompilovat běžnými C++ kompilátory pod MS Windows a nebo pomocí gcc 3.3.2 pod Linuxem. Současně s CxImage musí být slinkovány také tyto volně dostupné knihovny: zLib, LibTIFF, LibPNG, LibJPEG, JBIG-Kit, JasPer, LibJ2K.

Funkčnost knihovny je možné otestovat v aplikaci s názvem Demo, která je také součástí celého projektu. Aplikace obsahuje téměř všechny nástroje obsažené v knihovně. Výhodou je možnost pracovat s více otevřenými obrazy najednou, nevýhodou je pak to, že obrázek je otevřen pouze jednou a uživatel nemůže porovnat

upravený a původní obrázek. Čas potřebný k vykonání příslušné operace je zobrazován na stavovém řádku.

5.3 Časová náročnost vybraných algoritmů

Jako zdroj vstupních dat byla použita barevná fotografie o rozlišení 2592x1944 pixelů. Pro otestování některých funkcí byla převedena do odstínů šedi, a to z důvodu zajištění stejných podmínek pro všechny knihovny, protože některé operace v knihovně Filters provádějí tuto konverzi automaticky. Vstupní obraz byl dále prahován, protože morfologické operace implementované v knihovně PImage nepodporují jiné než binární obrazy.

Testování proběhlo na osobním počítači, s operačním systémem MS Windows XP, procesorem Intel Celeron 2.40GHz, 512MB RAM. K měření časů potřebných k vykonání jednotlivých metod z knihovny PImage bylo použito funkce *GetTickCount(),* která vrací čas v milisekundách od okamžiku spuštění systému. U ostatních knihoven bylo měření provedeno pomocí jejich vlastních testovacích programů. V tabulce (2) jsou uvedeny průměrné časy jednotlivých operací, každé měření bylo provedeno třikrát.

	barevný obraz			
	normalizace	ekvalizace	převzorkování[1)]	prahování
Filters				353
CxImage	5445	4846	53424	1104
PImage	1410	1141	15984	469

	černobílý obraz					binární obraz	
	Gaussian[2)]	medián[2)]	Průměr[2)]	Laplaceův op.	Sobelův op.	eroze[3)]	dilatace[3)]
Filters		4160	1207	1507	1143	2000	1971
CxImage	1782	23371				10412	10325
PImage	3347	4717	1994	540	1243	250	286

Tabulka 2 – Časové náročnosti (v milisekundách) jednotlivých algoritmů

Z tabulky je vidět, že operace prahování, filtrační metody (kromě Gaussiánu) a Sobelův hranový operátor jsou nejlépe implementovány v knihovně Filters. Naopak u Laplaceova hranového operátoru dosahuje lepších výsledků knihovna PImage. Velký rozdíl je dále v implementacích morfologických operací (dilatace, eroze), kde je PImage o řád rychlejší než Filters a téměř o dva řády rychlejší než CxImage. Velké rozdíly v časech jsou zřejmě způsobeny robustností implementace, protože obě knihovny (Filters, CxImage) podporují i barevné obrazy při morfologických operacích.

Modifikace jasové stupnice na základě histogramu (normalizace, ekvalizace) jsou zhruba čtyřikrát rychlejší u PImage než u CxImage. Nejrychlejší implementaci Gaussova filtru pak obsahuje CxImage, která však u všech ostatních operací dosahuje o řád horších výsledků. To může být způsobeno např. nevhodným použitím exportovaných funkcí v testovací aplikaci Demo.

Hodnoty v tabulce (2) jsou uvedeny v milisekundách. Některé hodnoty zde nejsou uvedeny z toho důvodu, že knihovna příslušné operace neobsahuje.

[1)] Obraz byl převzorkován na rozlišení 4000x3000 pixelů, bikubická interpolace.

[2)] Filtrace byli provedeny na okolí 3x3 body.

[3)] Při morfologických operacích byl použit čtvercový strukturní element, viz. (Obr. 1.7 c)

6. Závěr

Diplomová práce pojednává o technikách předzpracování a segmentace obrazu. Popisuje základní metody jako jsou například jasové korekce, modifikace jasové stupnice a geometrické transformace. Dále se zabývá různými druhy filtrací, jednoduchými i pokročilejšími hranovými detektory a matematickou morfologií binárních obrazů. Základní segmentační metoda prahování je uvedena hned v několika modifikacích. Sledování hranice, Houghova transformace a narůstání oblastí nakonec uzavírají tuto teoretickou část. Práce nepopisuje postupy zpracování obrazu jen z teoretického hlediska, ale snaží se i ukázat postup při jejich implementaci. Programování dynamických knihoven je věnována další část, která popisuje způsoby linkování a jejich výhody.

Dle zadání byla také vytvořena DLL knihovna PImage a testovací aplikace ImageProc, a to v prostředí vývojovém Microsoft Visual Studio .NET 2003. PImage obsahuje vybrané algoritmy předzpracování a segmentace v podobě exportovaných funkcí. Knihovna je v praxi použitelná a díky svému neobjektovému rozhraní i přenositelná do jiných vývojových prostředí. Jako další výhodu lze uvést nezávislost na jakýchkoli jiných prostředcích. Některé metody se podařilo s ohledem na časovou náročnost celkem efektivně implementovat, jak ostatně dokazuje kapitola 5. Ostatní algoritmy vykazují průměrné výsledky. Výjimkou je pouze Houghova transformace, která byla realizována s řadou omezení, a tvoří proto spíše jakýsi doplněk. Zajisté by se zde našel prostor i pro další optimalizace a vylepšení. Jednou z možností by mohlo být obohacení knihovny o další metody segmentace, popřípadě rozšíření nabídky podporovaných grafických formátů.

Ukázková aplikace slouží hlavně pro předvedení exportovaných funkcí, proto při její tvorbě nebyl příliš kladen důraz na použitelnost v praxi. Uživatel má možnost volat jednotlivé operace a současně sledovat co se děje s obrazem. Čas potřebný pro vykonání příslušné funkce je zobrazován na stavovém řádku.

Použité prameny

[1] Václav Hlaváč, Milan Šonka
Počítačové vidění
Grada, Praha, 1992, ISBN: 80-85424-67-3

[2] Václav Hlaváč, Miloš Sedláček
Zpracování signálů a obrazů
Skriptum ČVUT, ISBN: 80-01-02114-9

[3] Jiří Žára, Bedřich Beneš, Petr Felkel
Moderní počítačová grafika. 2. vydání
Computer Press, Praha, 1998, ISBN: 80-251-0454-0

[4] Radek Chaloupka
1001 tipů a triků pro Microsoft Visual C++
Computer Press, Brno, 2003, ISBN: 80-7226-842-2

[5] David J. Kruglinski, George Shepherd, Scot Wingo
Programujeme v Microsoft Visual C++
Computer Press, Brno, 2000, ISBN: 8072263625

[6] Michal Španěl, Vítězslav Beran
Obrazové segmentační techniky [online]
Vysoké učení technické v Brně, 2005, Dostupné z:
<http://www.fit.vutbr.cz/~spanel/segmentace/.en.iso-8859-2>

[7] Jiří Hořička
Počítačové zpracování digitálních obrázků – Houghova transformace. [online]
K^7 – vědecko populární časopis Fakulty mechatroniky TU v Liberci, 04/2005, s. 13 – 19, Dostupné z: <http://k7.vslib.cz/download/k7_05_4.pdf>,
ISSN: 1214-7370

[8] Alan Peters
Lectures on Image Processing [online]
Vanderbilt University, 1999-2007, Dostupné z:
<http://www.archive.org/details/Lectures_on_Image_Processing>

[9] Pavel Tišnovský
Seriál Grafické formáty [online]
2006, Dostupné z: <http://www.root.cz/serialy/graficke-formaty/>

[10] *Filters* [online]
2005 – 2007, Dostupné z: <http://filters.sourceforge.net/index.html>

[11] Davide Pizzolato
CxImage [online]
2003-2004, Dostupné z: <http://www.xdp.it/cximage.htm>

Přílohy

Příloha č.1. – přiložené cd, které obsahuje:

- spustitelnou verzi programu ImageProc a knihovny PImage
- všechny zdrojové kódy
- testovací obrázky
- text této zprávy

Příloha č.2. – výpis souboru *Interface.h*, který tvoří rozhraní knihovny PImage

```
struct _TPixels32
{
      BYTE  B;    //popr. U
      BYTE  G;    //V
      BYTE  R;    //Y
      BYTE  X;
}TPixels32, *PTPixels32;

typedef struct _TCImage
{
      PBYTE Data8, m_pPxs;
      _TPixels32 *Data32;
      RGBQUAD *pTab;
      unsigned char Priznak;
      BITMAPINFOHEADER InfoHead;
      BITMAPFILEHEADER FileHead;
      int *histogram;
}CImage;

//prototypy exportovanych fci
typedef BOOL (*TLoadFromFile)(char *szJmeno, CImage *pDest);
typedef BOOL (*TSaveToFile)(char *szJmeno, CImage *pSrc, int
nQuality);
typedef void (*TVykresli)(HDC hDC,HWND hMuj, CImage *Src);
typedef BOOL (*TDoCB)(CImage *pSrc);
typedef BOOL (*TNegativ)(CImage *pSrc);
typedef BOOL (*TFiltrPrumer)(CImage *pSrc, int n);
typedef BOOL (*TFiltrMed)(CImage *pSrc, int n);
typedef BOOL (*TFiltrRotMask)(CImage *pSrc, int n, BOOL median);
typedef BOOL (*TFiltrGauss)(CImage *pSrc, int n);
typedef BOOL (*THranyRoberts)(CImage *pSrc);
typedef BOOL (*THranySobel)(CImage *pSrc, int smer);
typedef BOOL (*THranyLaplace)(CImage *pSrc, int maska);
typedef BOOL (*THranyLoG)(CImage *pSrc, double sigma1, double sigma2);
typedef BOOL (*TCopyFromOrig)(CImage *pDest, BOOL co);
typedef BOOL (*TDestroy)(CImage *pSrc);
typedef BOOL (*TPrahovani)(CImage *pSrc, int prah);
typedef BOOL (*TProcPrahovani)(CImage *pSrc,int prah);
typedef void (*TSpustDialogHist)(HWND hWnd, CImage *pSrc);
typedef void (*TSpustDialogEl)(HWND hwnd);
typedef BOOL (*TNormalizaceJasu)(CImage *pSrc);
typedef void (*THistogram)(CImage *pSrc);
typedef BOOL (*TEkvalizaceHist)(CImage *pSrc);
typedef BOOL (*TDilatace)(CImage *pSrc, BYTE maska);
typedef BOOL (*TEroze)(CImage *pSrc, BYTE maska);
typedef BOOL (*TResample)(CImage *pDest,INT nW, INT nH);
typedef BOOL (*TSledHranice)(CImage *pSrc, int s);
```

```
typedef BOOL (*THoughTrans)(CImage *pSrc, int krivka,unsigned int
minR, unsigned int maxR, int okoli, double faktor );
typedef BOOL (*TNarOblasti)(CImage *pSrc, int krit);
typedef BOOL (*TBarPrahovani)(CImage *pSrc, int Rp, int Gp, int Bp);
typedef BOOL (*TPrahovaniInt)(CImage *pSrc, int dolni, int horni);
```